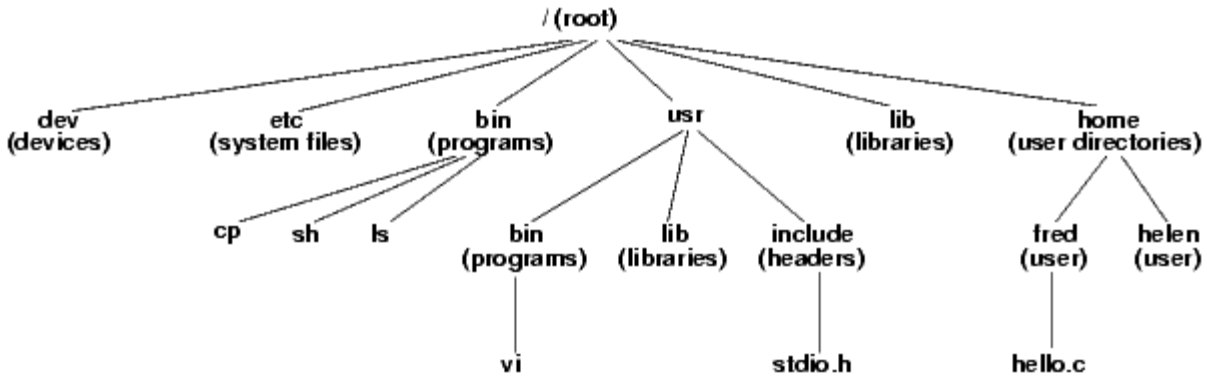
# UNIT-1

**The File System**

The UNIX file system looks like an inverted tree structure. You start with the **root** directory, denoted by /, at the top and work down through sub-directories underneath it.




Each node is either a **file** or a **directory** of files, where the latter can contain other files and directories. You specify a file or directory by its **path name**, either the full, or absolute, path name or the one relative to a location. The full path name starts with the root, /, and follows the branches of the file system, each separated by /, until you reach the desired file,

e.g.:

/home/condron/source/xntp

A relative path name specifies the path relative to another, usually the current working directory that you are at. Two special directory entries should be introduced now:

**.** the current directory

**..** the parent of the current directory

**Vi Editor:**

There are many ways to edit files in UNIX and for me one of the best ways is using screen-oriented text editor **vi**. This editor enables you to edit lines in context with other lines in the file.

Now a days you would find an improved version of vi editor which is called **VIM**.

Here VIM stands for **V**i **IM**proved.

The vi is generally considered the de facto standard in Unix editors because −

- It's usually available on all the flavors of Unix system.
- Its implementations are very similar across the board.
- It requires very few resources.
- It is more user friendly than any other editors like ed or ex.

You can use **vi** editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

Starting the vi Editor

There are following way you can start using vi editor −

| Command | Description |
|---|---|
| **vi filename** | Creates a new file if it already does not exist, otherwise opens existing file. |
| **vi -R filename** | Opens an existing file in read only mode. |
| **view filename** | Opens an existing file in read only mode. |

Following is the example to create a new file **testfile** if it already does not exist in the current working directory −

$vi testfile

As a result you would see a screen something like as follows −

```
|
~
~
~
"testfile" [New File]
```

You will notice a tilde (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other nonviewable character present.

So now you have opened one file to start with. Before proceeding further let us understanding few minor but important concepts explained below.

Operation Modes

While working with vi editor you would come across following two modes −

- **Command mode** − This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it is put in the file .

The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode you simply type **i**. To get out of insert mode, press the **Esc** key, which will put you back into command mode.

**Hint** − If you are not sure which mode you are in, press the Esc key twice, and then you'll be in command mode. You open a file using vi editor and start type some characters and then come in command mode to understand the difference.

Getting Out of vi

The command to quit out of vi is :q. Once in command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or :wq and return.

The easiest way to save your changes and exit out of vi is the **ZZ**command. When you are in command mode, type ZZ and it will do the equivalent of :wq.

Basic Commands in Unix:

who, date, stty, pwd, cd, mkdir, rmdir, ls, cp, mv, rm, cat, more, wc,

unlink, chmod ,ps, du, df, mount, umount, find, umask,unmask, ulimit, , w, finger, tail, head , sort,
nl, uniq, grep, egrep, fgrep, cut, paste, join, tee, pg, comm, cmp, diff, tr, tar, gzip,cpio
Who:

**DIRECTORY RELATED UTILITES:**

**1.Pwd**: Print the full pathname of the current directory.
Syntax: pwd
(Command name stands for "print working directory.")

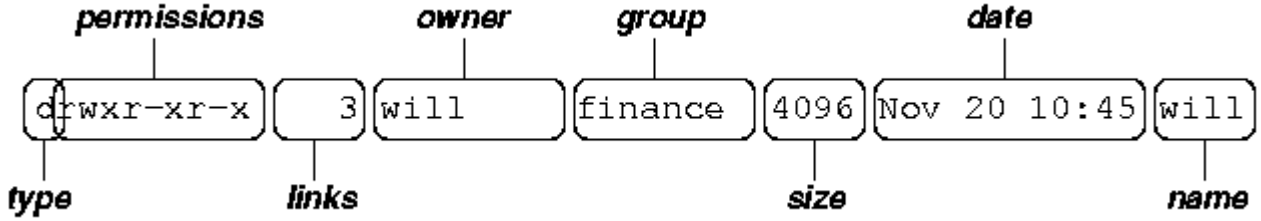
**2. mkdir:** Create one or more directories.
Syntax:
mkdir [options] directories
Example:
mkdir work;
mkdir junk;

**3. cd:** Change directory. cd is a built-in shell command

**4. rmdir** rmdir [options] directories
Delete the named directories (the directory itself, not the contents). directories are deleted from the parent directory and must be empty

**5.ls**
ls [options] [names]
If no names are given, list the files in the current directory. With one or more names, list files contained in a directory name or that match a file name. The options let you display a variety of information in different formats.
Options
-a :List all files, including the normally hidden . files.
-b :Show nonprinting characters in octal.
-c :List files by inode modification time.
-C :List files in columns (the default format, when displaying to a terminal device).
-d :List only the directory's information, not its contents. (Most useful with -l and -i.)
-f :Interpret each name as a directory (files are ignored).
-g :Like -l, but omit owner name (show group).
-i :List the inode for each file.
-l :Long format listing (includes permissions, owner, size, modification time, etc.).
-L :List the file or directory referenced by a symbolic link rather than the link itself.
-m :Merge the list into a comma-separated series of names.
-n :Like -l, but use user ID and group ID numbers instead of owner and group names.
-o :Like -l, but omit group name (show owner).
-p :Mark directories by appending / to them.
-q :Show nonprinting characters as ?.
-r :List files in reverse order (by name or by time).
-R :Recursively list subdirectories as well as current directory.
-s :Print sizes of the files in blocks.
-t :List files according to modification time (newest first).
-u :List files according to the file access time.
-x :List files in rows going across the screen.
-1 :Print one entry per line of output.
Examples
List all files in the current directory and their sizes; use multiple columns and mark special files:

ls -asCF
who or w
report who is logged in and what processes are running
**Eight Fields of ls –l:**




where:

- *type* is a single character which is either 'd' (directory), '-' (ordinary file), 'l' (symbolic link), 'b' (block-oriented device) or 'c' (character-oriented device).
- *permissions* is a set of characters describing access rights. There are 9 permission characters, describing 3 access types given to 3 user categories. The three access types are read ('r'), write ('w') and execute ('x'), and the three users categories are the user who owns the file, users in the group that the file belongs to and other users (the general public). An 'r', 'w' or 'x' character means the corresponding permission is present; a '-' means it is absent.
- *links* refers to the number of filesystem links pointing to the file/directory (see the discussion on hard/soft links in the next section).
- *owner* is usually the user who created the file or directory.
- *group* denotes a collection of users who are allowed to access the file according to the group access rights specified in the permissions field.
- *size* is the length of a file, or the number of bytes used by the operating system to store the list of files in a directory.
- *date* is the date when the file or directory was last modified (written to). The `-u` option display the time when the file was last accessed (read).
- *name* is the name of the file or directory.

## FILE HANDLING AND TEXT PROCESSING UTILITIES:

### cp - copy a file

Copy the contents of one file to another with the ***cp*** command.
**Syntax**
***cp*** [options] old_filename new_filename
**Common Options**
**-i** interactive (prompt and wait for confirmation before proceeding)

**-r** recursively copy a directory

**Examples**

cp old_filename new_filename

You now have two copies of the file, each with identical contents. They are completely independent of each other and you can edit and modify either as needed. They each have their own inode, data blocks, and directory table entries.

**mv - move a file**

Rename a file with the move command, ***mv***.

**Syntax**

***mv*** [options] old_filename new_filename

**Common Options**

**-i** interactive (prompt and wait for confirmation before proceeding)

**-f** don't prompt, even when copying over an existing target file (overrides **-i**)

**Examples**

mv old_filename new_filename

You now have a file called **new_filename** and the file **old_filename** is gone. Actually all you've

done is to update the directory table entry to give the file a new name. The contents of the file remain where they were.

**rm - remove a file**

Remove a file with the ***rm***, remove, command.

**Syntax**

***rm*** [options] filename

**Common Options**

**-i** interactive (prompt and wait for confirmation before proceeding)

**-r** recursively remove a directory, first removing the files and subdirectories

beneath it

**-f** don't prompt for confirmation (overrides -i)

**Examples**

rm old_filename

A listing of the directory will now show that the file no longer exists. Actually, all you've done is to remove the directory table entry and mark the inode as unused. The file contents are still on the disk,but the system now has no way of identifying those data blocks with a file name.

**File Permissions**

Each file, directory, and executable has permissions set for who can **read**, write, and/or **execute** it.

To find the permissions assigned to a file, the ***ls*** command with the **-l** option should be used. Also, using the **-g** option with "***ls -l***" will help when it is necessary to know the group for which the permissions are

When using the "***ls -l***" command on a file the output will appear as follows:

-rwxr-x--- user unixgroup size Month nn hh:mm filename

The area above designated by letters and dashes (**-rwxr-x---**) is the area showing the file type andpermissions as defined in the previous Section. Therefore, a permission string, for example, of-**rwxr-x---** allows the **user** (owner) of the file to read, write, and execute it; those in the **unixgroup** of the file can read and execute it; **others** cannot access it at all.

**chmod - change file permissions**

The command to change permissions on an item (file, directory, etc) is ***chmod*** (change mode). The syntax involves using the command with three digits (representing the **user**

(owner, **u**) permissions, the **group** (**g**) permissions, and **other** (**o**) user's permissions) followed by the argument (which may be a file name or list of files and directories). Or by using symbolic representation for the permissions and who they apply to.
Each of the permission types is represented by either a numeric equivalent:
read=4, write=2, execute=1 or a single letter:
read=r, write=w, execute=x
A permission of **4** or **r** would specify **read** permissions. If the permissions desired are read and write,the 4 (representing read) and the 2 (representing write) are added together to make a permission of 6.Therefore, a permission setting of 6 would allow read and write permissions.
Alternatively, you could use symbolic notation which uses the one letter representation for who and for the permissions and an operator, where the operator can be:
+ add permissions - remove permissions = set permissions
So to set read and write for the owner we could use "**u=rw**" in symbolic notation.
**Syntax**
***chmod*** nnn [argument list] numeric mode
***chmod*** [who]op[perm] [argument list] symbolic mode
where **nnn** are the three numbers representing **user**, **group**, and **other** permissions, **who** is any of **u**, **g**,**o**, or **a** (all) and **perm** is any of **r**, **w**, **x**. In symbolic notation you can separate permission
specifications by commas, as shown in the example below.
**Common Options**
**-f** force (no error message is generated if the change is unsuccessful)
**-R** recursively descend through the directory structure and change the modes
**Examples**
If the permission desired for file1 is **user**: read, write, execute, **group**: read, execute, **other**: read,
execute, the command to use would be
***chmod 755 file1*** or ***chmod u=rwx,go=rx file1***
**Reminder**: When giving permissions to **group** and **other** to use a file, it is necessary to allow at leastexecute permission to the directories for the path in which the file is located. The easiest way to do this is to be in the directory for which permissions need to be granted:
***chmod 711 .*** or ***chmod u=rw,+x .*** or ***chmod u=rwx,go=x .***
where the dot (**.**) indicates **this directory**.

**chown - change ownership**
Ownership of a file can be changed with the ***chown*** command. On most versions of Unix this can
only be done by the super-user, i.e. a normal user can't give away ownership of their files. ***chown*** is used as below, where **#** represents the shell prompt for the super-user:
**Syntax**
***chown*** [options] user[:group]
**Common Options**
**-R** recursively descend through the directory structure
**-f** force, and don't report any errors
**Examples**
# chown new_owner file
**chgrp - change group**
Anyone can change the group of files they own, to another group they belong to, with the ***chgrp***

command.

**Syntax**

***chgrp*** [options] group file

**Common Options**

**-R** recursively descend through the directory structure

**-f** force, and don't report any errors

**Examples**

% chgrp new_group file

**echo - echo a statement**

The ***echo*** command is used to repeat, or echo, the argument you give it back to the standard output device. It normally ends with a line-feed, but you can specify an option to prevent this.

**Syntax**

***echo*** [string]

**Common Options**

**-n** don't print <new-line>

**\c** don't print <new-line>

**\t** tab

**\f** form-feed (SVR4)

**\n** new-line (SVR4)

**\v** vertical tab (SVR4)

**Examples**

% echo Hello Class or echo "Hello Class"

To prevent the line feed:

% echo -n Hello Class or echo "Hello Class \c"

where the style to use in the last example depends on the ***echo*** command in use.

The \x options must be within pairs of single or double quotes, with or without other string characters.

**cat - concatenate a file**

Display the contents of a file with the concatenate command, ***cat***.

**Syntax**

***cat*** [options] [file]

**Common Options**

**-n** precede each line with a line number

**-v** display non-printing characters, except tabs, new-lines, and form-feeds

**-e** display $ at the end of each line (prior to new-line) (when used with **-v** option)

**Examples**

% cat filename

You can list a series of files on the command line, and ***cat*** will concatenate them, starting each in turn, immediately after completing the previous one,

e.g.:

cat file1 file2 file3

**more, less, and pg - page through a file**

***more***, ***less***, and ***pg*** let you page through the contents of a file one screenful at a time. These may not all be available on your Unix system. They allow you to back up through the previous pages and search for words, etc.

**Syntax**

***more*** [options] [+/pattern] [filename]

***less*** [options] [+/pattern] [filename]

***pg*** [options] [+/pattern] [filename]

**Options**

**more less pg Action**
**-c -c -c** clear display before displaying
**-i** ignore case
**-w** default default don't exit at end of input, but prompt and wait
**-lines -lines** # of lines/screenful
**+/pattern +/pattern +/pattern** search for the pattern
**head - display the start of a file**
***head*** displays the head, or start, of the file.
**Syntax**
***head*** [options] file
**Common Options**
**-n** number number of lines to display, counting from the top of the file
**-number** same as above
**Examples**
By default ***head*** displays the first 10 lines. You can display more with the "**-n numbe**r", or "**-number**" options, e.g., to display the first 40 lines:
$ head -40 filename or head -n 40 filename
**tail - display the end of a file**
***tail*** displays the tail, or end, of the file.
**Syntax**
***tail*** [options] file
**Common Options**
**-number** number of lines to display, counting from the bottom of the file
**Examples**
The default is to display the last 10 lines, but you can specify different line or byte numbers, or a
different starting point within the file. To display the last 30 lines of a file use the **-number** style:
$ tail -30 filename
**date - current date and time**
***date*** displays the current data and time. A superuser can set the date and time.
**Syntax**
***date*** [options] [+format]
**Common Options**
**-u** use Universal Time (or Greenwich Mean Time)
**+format** specify the output format
**%a** weekday abbreviation, Sun to Sat
**%h** month abbreviation, Jan to Dec
**%j** day of year, 001 to 366
**%n** <new-line>
**%t** <TAB>
**%y** last 2 digits of year, 00 to 99
**%D** MM/DD/YY date
**%H** hour, 00 to 23
**%M** minute, 00 to 59
**%S** second, 00 to 59
**%T** HH:MM:SS time
**Examples**
beauty condron>date
Mon Jun 10 09:01:05 EDT 1996

beauty condron>date -u
Mon Jun 10 13:01:33 GMT 1996
beauty condron>date +%a%t%D
Mon 06/10/96
beauty condron>date '+%y:%j'
96:162

**cmp - compare file contents**

The ***cmp*** command compares two files, and (without options) reports the location of the first difference between them. It can deal with both binary and ASCII file comparisons. It does a byte-by-byte comparison.

**Syntax**

***cmp*** [options] file1 file2 [skip1] [skip2]

The **skip** numbers are the number of bytes to skip in each file before starting the comparison.

**Common Options**

**-l** report on each difference

**-s** report exit status only, not byte differences

**diff - differences in files**

The ***diff*** command compares two files, directories, etc, and reports all differences between the two. It deals only with ASCII files. It's output format is designed to report the changes necessary to convert the first file into the second.

**Syntax**

***diff*** [options] file1 file2

**Common Options**

**-b** ignore trailing blanks

**-i** ignore the case of letters

**-w** ignore <**space**> and <**tab**> characters

**-e** produce an output formatted for use with the editor, ***ed***

**-r** apply diff recursively through common sub-directories

**cut - select parts of a line**

The ***cut*** command allows a portion of a file to be extracted for another use.

**Syntax**

***cut*** [options] file

**Common Options**

**-c** character_list character positions to select (first character is 1)

**-d** delimiter field delimiter (defaults to <**TAB**>)

**-f** field_list fields to select (first field is 1)

Both the character and field lists may contain comma-separated or blank-character-separated numbers (in increasing order), and may contain a hyphen (-) to indicate a range. Any numbers

missing at either before (e.g. -5) or after (e.g. 5-) the hyphen indicates the full range starting with thefirst, or ending with the last character or field, respectively. Blank-character-separated lists must beenclosed in quotes. The field delimiter should be enclosed in quotes if it has special meaning to the shell, e.g. when specifying a <**space**> or **<TAB>** character.

**Examples**

In these examples we will use the file **users**:

jdoe John Doe 4/15/96
lsmith Laura Smith 3/12/96
pchen Paul Chen 1/5/96
jhsu Jake Hsu 4/17/96

sphilip Sue Phillip 4/2/96
If you only wanted the username and the user's real name, the ***cut*** command could be used to get only
that information:
% cut -f 1,2 users
jdoe John Doe
lsmith Laura Smith
pchen Paul Chen
jhsu Jake Hsu
sphilip Sue Phillip

**paste - merge files**

The ***paste*** command allows two files to be combined side-by-side. The default delimiter between the columns in a paste is a tab, but options allow other delimiters to be used.

**Syntax**

***paste*** [options] file1 file2

**Common Options**

**-d** list list of delimiting characters
**-s** concatenate lines

The list of **delimiters** may include a single character such as a comma; a quoted string, such as a
space; or any of the following escape sequences:
\n <newline> character
\t <tab> character
\\ backslash character
\0 empty string (non-null character)
It may be necessary to quote delimiters with special meaning to the shell.
A hyphen (**-**) in place of a file name is used to indicate that field should come from standard input.

**touch - create a file**

The touch command can be used to create a new (empty) file or to update the last access date/time onan existing file. The command is used primarily when a script requires the pre-existence of a file (for example, to which to append information) or when the script is checking for last date or time afunction was performed.

**Syntax *touch*** [options] [date_time] file

**wc - count words in a file**

***wc*** stands for "word count"; the command can be used to count the number of lines, characters, or
words in a file.

**Syntax**

***wc*** [options] file

**Common Options**

**-c** count bytes
**-m** count characters (SVR4)
**-l** count lines
**-w** count words

**ln - link to another file**

The ***ln*** command creates a "link" or an additional way to access (or gives an additional name to)
another file.

**Syntax**

***ln*** [options] source [target]
If not specified **target** defaults to a file of the same name in the present working directory.
**Common Options**
**-f** force a link regardless of target permissions; don't report errors (SVR4 only)
**-s** make a symbolic link
**Examples**
A **symbolic link** is used to create a new path to another file or directory
**sort - sort file contents**
The ***sort*** command is used to order the lines of a file. Various options can be used to choose the order
as well as the field on which a file is sorted. Without any options, the sort compares entire lines in the
file and outputs them in ASCII order (numbers first, upper case letters, then lower case letters).
**Syntax**
***sort*** [options] [+pos1 [ -pos2 ]] file

**Common Options**
**-b** ignore leading blanks (<**space**> & <**tab**>) when determining starting and
ending characters for the sort key
**-d** dictionary order, only letters, digits, <**space**> and <**tab**> are significant
**-f** fold upper case to lower case
**-k** keydef sort on the defined keys (not available on all systems)
**-i** ignore non-printable characters
**-n** numeric sort
**-o** outfile output file
**-r** reverse the sort
**-t** char use char as the field separator character
**-u** unique; omit multiple copies of the same line (after the sort)
+pos1 [-pos2] (old style) provides functionality similar to the "-k keydef" option.
**tee - copy command output**
***tee*** sends standard in to specified files and also to standard out. It's often used in command pipelines.
**Syntax**
***tee*** [options] [file[s]]
**Common Options**
**-a** append the output to the files
**-i** ignore interrupts
**uniq - remove duplicate lines**
***uniq*** filters duplicate adjacent lines from a file.
**Syntax**
***uniq*** [options] [+|-n] file [file.new]
**Common Options**
**-d** one copy of only the repeated lines
**-u** select only the lines not repeated
**+n** ignore the first **n characters**
**-s** n same as above
**-n** skip the first **n fields**, including any blanks (<**space**> & <**tab**>)
**-f** fields same as above
**tr - translate characters**

The ***tr*** command translates characters from stdin to stdout.

**Syntax**

***tr*** [options] string1 [string2]

With no options the characters in **string1** are translated into the characters in **string2**, character by character in the string arrays. The first character in **string1** is translated into the first character in **string2**, etc.

To translate all lower case alphabetic characters to upper case we could use either of:

tr '[a-z]' '[A-Z]' or tr '[:lower:]' '[:upper:]'

**FILE ARCHIVING, COMPRESSION AND CONVERSION COMMANDS:**

## File Compression

The ***compress*** command is used to reduce the amount of disk space utilized by a file. When a file has been compressed using the ***compress*** command, a suffix of **.Z** is appended to the file name. The ownership modes and access and modification times of the original file are preserved. ***uncompress*** restores the files originally compressed by ***compress***.

**Syntax**

***compress*** [options] [file]

***uncompress*** [options] [file.Z]

***zcat*** [file.Z]

**Common Options**

**-c** write to standard output and don't create or change any files

**-f** force compression of a file, even if it doesn't reduce the size of the file or if the target file (file.Z) already exists.

**-v** verbose. Report on the percentage reduction for the file.

The GNU programs to provide similar functions to those above are often installed as ***gzip***, ***gunzip***, and ***zcat***, respectively. Files compressed with gzip are given the endings **.z** or **.gz**.

## tar - archive files

The ***tar*** command combines files into one device or filename for archiving purposes. The ***tar*** command does not compress the files; it merely makes a large quantity of files more manageable.

**Syntax**

***tar*** [options] [directory file]

**Common Options**

**c** create an archive (begin writting at the start of the file)

**t** table of contents list

**x** extract from an archive

**v** verbose

**f** archive file name

**b** archive block size

***tar*** will accept its options either with or without a preceding hyphen (**-**). The archive file can be a disk file, a tape device, or standard input/output. The latter are represented by a hyphen.

**SYSTEM RESOURCE COMMANDS:**

## df - summarize disk block and file usage

***df*** is used to report the number of disk blocks and inodes used and free for each file system. The output format and valid options are very specific to the OS and program version in use.

**Syntax**

***df*** [options] [resource]

**Common Options**

**-l** local file systems only
**-k** report in kilobytes

## du - report disk space in use

***du*** reports the amount of disk space in use for the files or directories you specify.

**Syntax**

***du*** [options] [directory or file]

**Common Options**

**-a** display disk usage for each file, not just subdirectories
**-s** display a summary total only
**-k** report in kilobytes

## ps - show status of active processes

***ps*** is used to report on processes currently running on the system. The output format and valid options are very specific to the OS and program version in use.

**Syntax**

***ps*** [options]

**Mount:**

A file system must be mounted in order to be usable by the system. To see what is currently mounted (available for use) on your system, use this command:
$mount

**Syntax:**

mount -t file_system_type device_to_mount directory_to_mount_to
For example, if you want to mount a CD-ROM to the directory /mnt/cdrom, for example, you can type:
$ mount -t iso9660 /dev/cdrom /mnt/cdrom
This assumes that your CD-ROM device is called /dev/cdrom and that you want to mount it to /mnt/cdrom. Refer to the mount man page for more specific information or type mount -h at the command line for help information.
After mounting, you can use the cd command to navigate the newly available file system through the mountpoint you just made.

**Unmounting the File System:**

To unmount (remove) the file system from your system, use the **umount** command by identifying the mountpoint or device
For example, to unmount cdrom, use the following command:
$ umount /dev/cdrom

## find - find files

The ***find*** command will recursively search the indicated directory tree to find files matching a type or pattern you specify. ***find*** can then list the files or execute arbitrary commands based on the results.

**Syntax**

***find*** directory [search options] [actions]

**Common Options**

For the time search options the notation in days, **n** is:
**+n** more than **n** days
**n** exactly **n** days
**-n** less than **n** days

**Umask** :

Show the permissions that are given to view files by default

**ulimit**

**NAME**

**ulimit - Modify shell resource limits.**

**SYNOPSIS**

ulimit [-SHacdefilmnpqrstuvx] [limit]

**DESCRIPTION**

**Modify shell resource limits.**

Provides control over the resources available to the shell and processes it creates, on systems that allow such control.

## finger - get information about users

***finger*** displays the **.plan** file of a specific user, or reports who is logged into a specific machine. The user must allow general read permission on the **.plan** file.

**Syntax**

***finger*** [options] [user[@hostname]]

**Common Options**

**-l** force long output format

**-m** match username only, not first or last names

**-s** force short output format

**cpio** copies files to and from archives. cpio stands for "copy in, copy out".

**TEXT PROCESSING COMMANDS:**

The grep program searches a file or files for lines that have a certain pattern. The syntax is:
$grep pattern file(s)

| Option | Description |
|---|---|
| -v | Print all lines that do not match pattern. |
| -n | Print the matched line and its line number. |
| -l | Print only the names of files with matching lines (letter "l") |
| -c | Print only the count of matching lines. |
| -i | Match either upper- or lowercase. |

Displays only Aug Files

$ls -l | grep "Aug"

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

**egrep** is the same as running **grep -E**. In this mode, **grep** evaluates your*PATTERN* string as an extended regular expression (ERE). Nowadays, ERE does not "extend" very far beyond basic regular expressions, but they can still be very useful.

fgrep is the same as running grep -F. In this mode, grep evaluates your PATTERNstring as a "fixed string" — every character in your string is treated literally. For example, if your string contains an asterisk ("*"), grep will try to match it with an actual asterisk rather than

interpreting this as a wildcard. If your string contains multiple lines (if it contains newlines), each line will be considered a fixed string, and any of them can trigger a match.

```
grep 'fred' /etc/passwd            # search for lines containing 'fred' in /etc/passwd
grep fred /etc/passwd              # quotes usually not when you don't use regex patterns
grep null *.scala                  # search multiple files
```

**case-insensitive**

```
grep -i joe users.txt              # find joe, Joe, JOe, JOE, etc.
```

**regular expressions**

```
grep '^fred' /etc/passwd           # find 'fred', but only at the start of a line
grep '[FG]oo' *                    # find Foo or Goo in all files in the current dir
grep '[0-9][0-9][0-9]' *           # find all lines in all files in the current dir with three numbers in a row
```

**display matching filenames, not lines**

```
grep -l StartInterval *.plist      # show all filenames containing the string 'StartInterval'
grep -il StartInterval *.plist     # same thing, case-insensitive
```

**show matching line numbers**

```
grep -n we gettysburg-address.txt  # show line numbers as well as the matching lines
```

**lines before and after grep match**

```
grep -B5 "the living" gettysburg-address.txt        # show all matches, and five lines before each match
grep -A10 "the living" gettysburg-address.txt       # show all matches, and ten lines after each match
grep -B5 -A5 "the living" gettysburg-address.txt    # five lines before and ten lines after
```

**reverse the meaning**

```
grep -v fred /etc/passwd           # find any line *not* containing 'fred'
grep -vi fred /etc/passwd          # same thing, case-insensitive
```

**grep in a pipeline**

```
ps auxwww | grep httpd             # all processes containing 'httpd'
ps auxwww | grep -i java           # all processes containing 'java', ignoring case
ls -al | grep '^d'                 # list all dirs in the current dir
```

**search for multiple patterns**

```
egrep 'apple|banana|orange' *                  # search for multiple patterns, all files in current dir
egrep -i 'apple|banana|orange' *               # same thing, case-insensitive
```

**multiple search strings, multiple filename patterns:**

```
grep -li "jtable" $(find . -name "*.java,v" -exec grep -li "prevayl" {} \;)        # find all files named "*.java,v" containing both # 'prevayl' and 'jtable'
```

**grep + find**

```
find . -type f -exec grep -il 'foo' {} \;      # print all filenames of files under current dir containing 'foo', case-insensitive
```

**recursive grep search**

```
grep -rl 'null' .                              # very similar to the previous find command; does a recursive search
grep -ril 'null' /home/al/sarah /var/www       # search multiple dirs
egrep -ril 'aja|alvin' .                       # multiple patterns, recursive
```